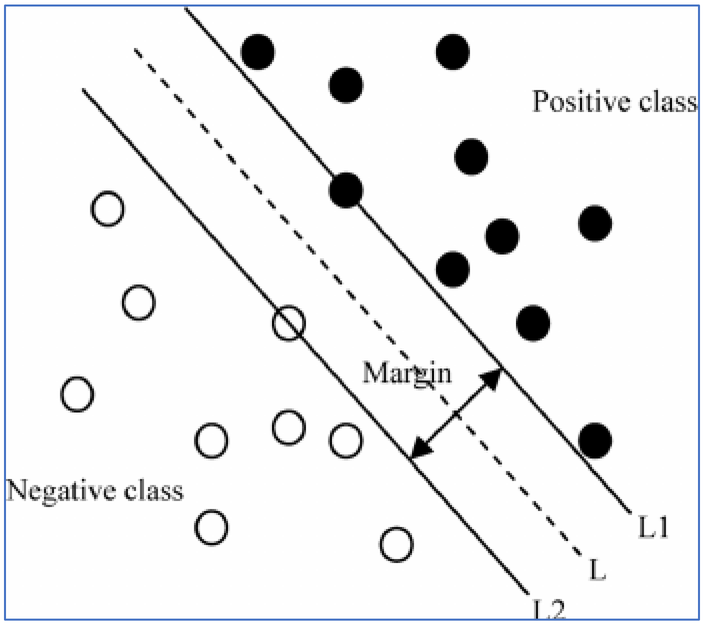
Kartik Chanana, Anirudh Devgun, Sameer Motwani

CS6240 (Section 1) – Final Project

We tested on 2 different models –

- Support Vector Machine
- Random Forest

Support Vector Machine: Support Vector Machine is a supervised machine learning algorithm for classification or regression problems where the dataset teaches SVM about the classes so that SVM can classify any new data. It works by classifying the data into different classes by finding a line (hyperplane) which separates the training data set into classes. As there are many such linear hyperplanes, SVM algorithm tries to maximize the distance between the various classes that are involved and this is referred as margin maximization. If the line that maximizes the distance between the classes is identified, the probability to generalize well to unseen data is increased.



Tasks for 100 iterations of SVM came out to be 108. We tested for multiple configurations and the below configurations helped us determine the performance –

Number of Iterations	**Reg Factor**	**Accuracy**	**Area Under ROC**
1	1	99.1479	0.5
50	1	99.1479	0.5
75	1	99.1479	0.5
100	1	99.1480	0.500179
50	0.1	99.1479	0.5

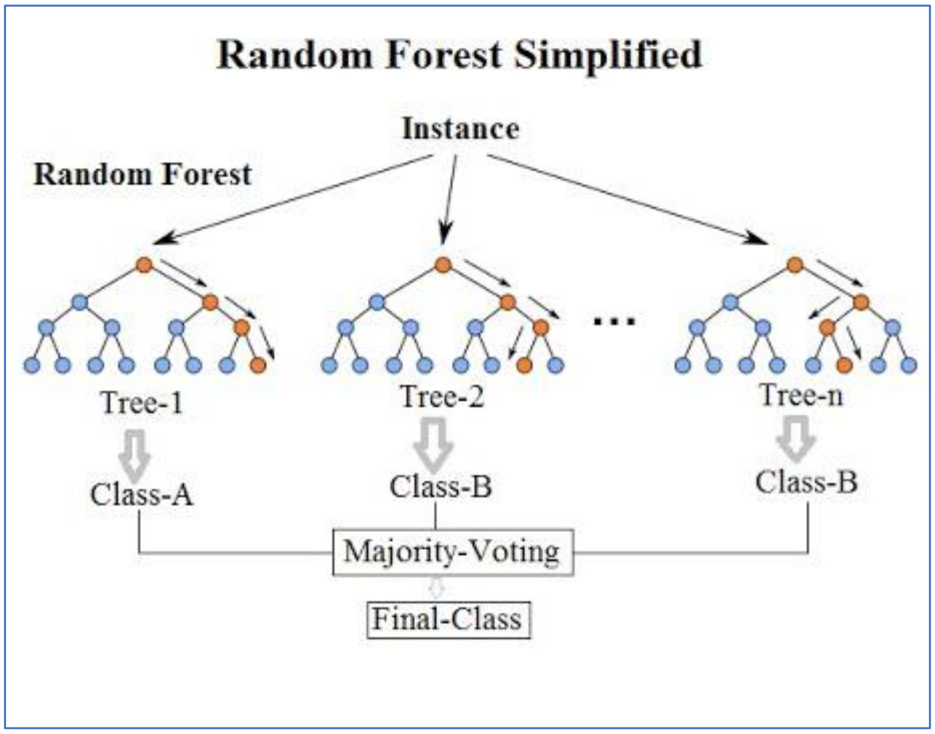
Random forest:

Random Forest is the go to machine learning algorithm that uses a bagging approach to create a bunch of decision trees with random subset of the data. A model is trained several times on random sample of the dataset to achieve good prediction performance from the random forest algorithm. In this ensemble learning method, the output of all the decision trees in the random forest, is combined to make the final prediction. The final prediction of the random forest algorithm is derived by polling the results of each decision tree or just by going with a prediction that appears the most times in the decision trees.

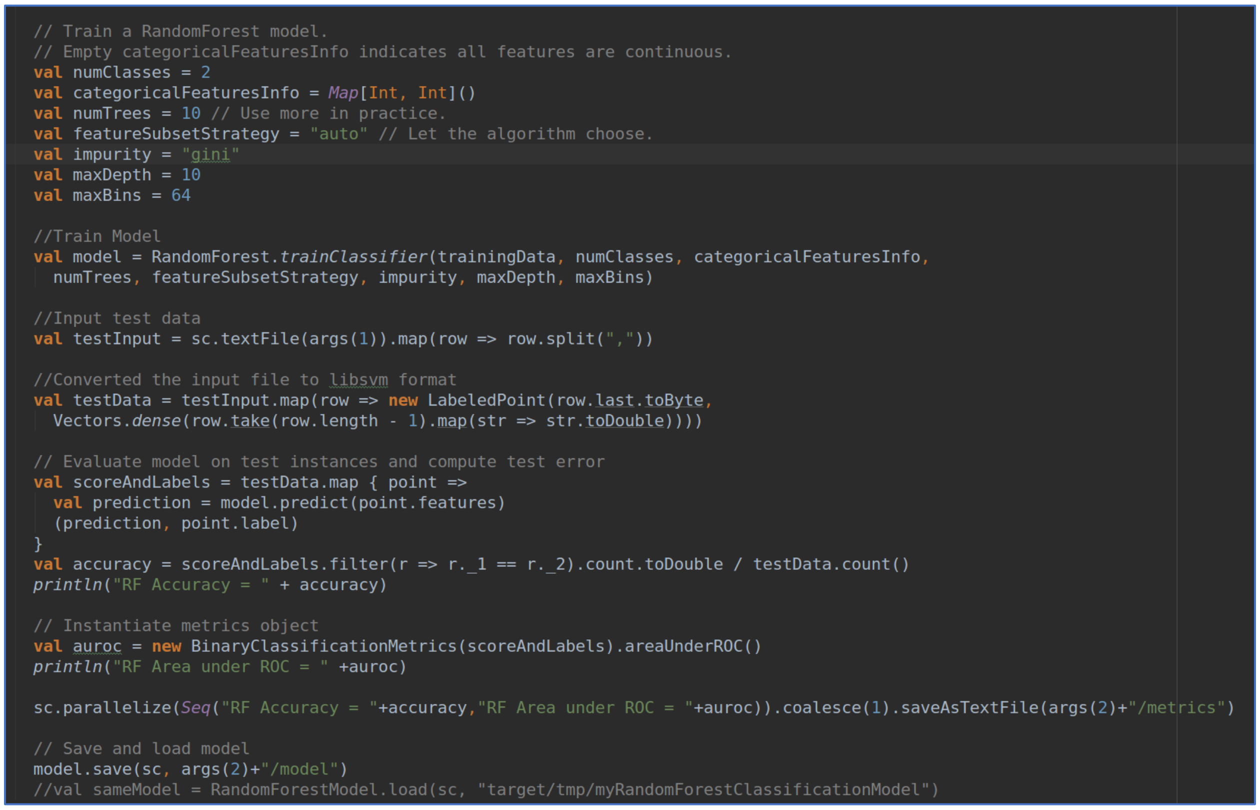
Advantages of Random Forest over other classification algorithms:

- It maintains accuracy when there is missing data and is also resistant to outliers.
- Simple to use as the basic random forest algorithm can be implemented with just a few lines of code. It maintains accuracy when there is missing data and is also resistant to outliers.
- Simple to use as the basic random forest algorithm can be implemented with just a few lines of code.
- Random Forest is one of the most effective and versatile machine learning algorithm for wide variety of classification and regression tasks, as they are more robust to noise.
- Random Forest machine learning algorithms can be grown in parallel.
- Has higher classification accuracy.

Because of the above-mentioned reasons, we chose Random forest as the model over SVM.



Pseudo code:



```
// Train a RandomForest model.
// Empty categoricalFeaturesInfo indicates all features are continuous.
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val numTrees = 10 // Use more in practice.
val featureSubsetStrategy = "auto" // Let the algorithm choose.
val impurity = "gini"
val maxDepth = 10
val maxBins = 64

//Train Model
val model = RandomForest.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo,
  numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)

//Input test data
val testInput = sc.textFile(args(1)).map(row => row.split(","))

//Converted the input file to libsvm format
val testData = testInput.map(row => new LabeledPoint(row.last.toByte,
  Vectors.dense(row.take(row.length - 1).map(str => str.toDouble))))

// Evaluate model on test instances and compute test error
val scoreAndLabels = testData.map { point =>
  val prediction = model.predict(point.features)
  (prediction, point.label)
}
val accuracy = scoreAndLabels.filter(r => r._1 == r._2).count.toDouble / testData.count()
println("RF Accuracy = " + accuracy)

// Instantiate metrics object
val auroc = new BinaryClassificationMetrics(scoreAndLabels).areaUnderROC()
println("RF Area under ROC = " +auroc)

sc.parallelize(Seq("RF Accuracy = "+accuracy,"RF Area under ROC = "+auroc)).coalesce(1).saveAsTextFile(args(2)+"/metrics")

// Save and load model
model.save(sc, args(2)+"/model")
//val sameModel = RandomForestModel.load(sc, "target/tmp/myRandomForestClassificationModel")
```

How many tasks are created during each stage of the model training process?

Ans:

Stage Description	**Tasks**
Take at decisionTreeMetadata	1
Count at decisionTreeMetadata	385
collectAsMap at RandomForestTrain.scala	457
Count at RandomForestTrain.scala	97
sortByKey at RandomForestTrain.scala	169
saveAsTextFile at RandomForestTrain.scala	72

How many iterations are executed during model training?

Ans: For SVM, we used different values of iterations for example 1, 50,75 and 100. Random forest doesn’t have multiple iterations.

Is the data being shuffled?

Ans: Yes the data is being shuffled when sortByKey is called.

How did the change of parameters controlling partitioning affect the running time?

Ans: As we increased the number of trees and the max depth parameter, the running time increased.

Preprocessing:

No we didn't do any preprocessing on the data given. However, we did, convert the input and the test data into an RDD of LabeledPoints in MLlib.

Accuracy numbers for different parameter settings:

Number of Trees	Max Depth	Number of Bins	Accuracy	Area Under ROC
10	4	32	99.5797	0.779993
5	4	32	99.5214	0.739926
10	10	32	99.7375	0.894846
5	10	32	99.6892	0.862126
20	10	32	99.7376	0.886115
20	5	32	99.6185	0.800823
30	10	32	99.7402	0.884453
10	10	64	99.7467	0.893935
10	20	32	99.7168	0.909515

As per the above table, we found that the accuracy was maximum when we set the number of trees to 10, max depth to 10 and number of bins 64.

Running Time:

Number of workers	Steps	Time Taken
5	Model Training	63 mins
	Prediction	7 mins
10	Model Training	28 mins
	Prediction	7 mins
15	Model Training	14 mins
	Prediction	7 mins